IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

In re Application of:

Applicant: Kosche et al.	
App. No.: 10/840,164	Conf. No.: 7512
Filed: May 6, 2004	Art Unit: 2192
Title: METHOD AND APPARATUS FOR PROFILING DATA ADDRESSES	Examiner: Tecklu, Isaac T.

APPEAL BRIEF

MAIL STOP APPEAL BRIEF - PATENTS
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Sir:

    Applicant (hereafter "Appellant") hereby submits this Appeal Brief in response to the to the final Office action mailed September 29, 2009 and the Notice of Appeal filed on January 28, 2010 in the above-captioned case.

    Appellant respectfully requests consideration of this appeal by the Board of Patent Appeals and Interferences (hereafter the "Board").

    An oral hearing is not requested at this time.

TABLE OF CONTENTS

I.   **REAL PARTY IN INTEREST**

The Appellant is Oracle America, Inc. of 500 Oracle Parkway, Redwood Shores, California 94065.

II.   **RELATED APPEALS AND INTERFERENCES**

To the best of Appellant's knowledge, there are no appeals or interferences that are related to, will directly affect, will be directly affected by, or have a bearing on the Board's decision in the present appeal.

III.   **STATUS OF THE CLAIMS**

Claims 1-3, 5-12, 14-27, 29-37, 39-42, 44-50, 52-56 and 58 are currently pending in the above-referenced application. The rejections of all of the pending claims are appealed herein. No claims have been allowed. Claims 4, 13, 28, 38, 43, 51 and 57 are cancelled. A clean copy of all claims on appeal is attached hereto as the Appendix of Claims. We note that previously cancelled claims are not provided in the Appendix.

IV.   **STATUS OF AMENDMENTS**

The last response submitted by the Appellant was dated December 8, 2009, no claims were amended in response to the final Office action dated September 9, 2009 (hereinafter the "final Office action"). An Advisory Action was entered January 4, 2010, to which no further response or amendment was submitted by the Appellant. A Notice of Appeal was filed on January 28, 2010.

V.   SUMMARY OF THE CLAIMED SUBJECT MATTER

In general, the systems and methods claimed allow for the profiling of computer code during runtime to identify portions of code that consume excessive amounts of execution time—i.e., runtime events. *See, e.g., specification page 3, lines 19-21.* The term "runtime event" describes events that occur during the execution of code that hinder execution, such as cache misses, cache references, data translation buffer misses, branch mispredicts, etc. *See, e.g., specification page 6, lines 18-20.* By profiling these runtime events, software developers can more efficiently optimize their code. *See, e.g., specification page 7, lines 15-16.* While conventional profiling tools provide per image, per procedure, per source line or per instruction level profile information, these tools do not provide profile information in relation to other aspects of code behavior. *See, e.g., specification page 3, lines 7-9.* In particular, the conventional profiling tools do not perform data profiling, such as memory related operations from the perspective of data objects, addresses of data objects, or data object definitions. *See, e.g., specification page 3, lines 9-15.* The majority of stall time within a program is caused by memory related operations or load type instructions instances, but conventional tools do not provide information about data objects that consume the most of amount of execution time. *Id.*

The claimed systems and methods, however, provide information regarding code behavior from the perspective of data addresses. *See, e.g., specification page 3, lines 19-22.* The addresses are also associated with memory reference objects (e.g., memory segments, heap variables, variable instances, stack variables, etc.) *See, e.g., specification page 3, lines 26-28.* Associating sampled runtime events with addresses allows code hindrances to be defined from different perspectives of memory references and allows for sampled runtime events to be correlated with memory reference objects. *See, e.g., specification page 3, lines 20-23.*

Independent claim 1 is directed to a computer-implemented software profiling tool that determines at least one data address from one or more instruction instances. *See e.g., FIG.4, reference numeral 419; See also, e.g., specification page 15, lines 20-25.* For each data address one or more memory references objects (associated with the data addresses), are identified as hindering execution of code that includes instruction instances. *See, e.g., specification page 16, lines 5-15.* The memory reference objects include virtually addressable memory. *Id.*

Independent claim 16 is directed to a method for profiling code executing in a computer. The instruction instances corresponding to a runtime event are identified and then a data address, including a virtual address, for the instruction instances is determined.

See FIG. 6, reference numeral 621. Then a memory reference object is determined from the determined address. *See also, e.g., specification page 15, lines 5-20.*

Independent claim 33 is directed to a method of profiling code executing in a computer. Data addresses, including virtual addresses, are associated with memory reference objects, where the data addresses have been determined from instruction instances corresponding to code execution hindrance. *See, e.g., specification page 16, lines 5-15.* Then data addresses are aggregated based on their associated memory reference objects. *See FIG. 10, reference numeral 1015; specification page 21, lines 20-30.*

Independent claim 41 is directed to a method of profiling code in a computer system. An instruction instance corresponding to a runtime event is identified and determined whether or not it is valid. *See, FIG. 4, reference numerals 409, 413.* The instruction instance is then decoded to extract at least a portion of a data address (if the instruction instance is valid). *See FIG. 4, reference numeral 417; specification page 14, lines 5-7.* A memory reference object, which includes virtual addresses in the computer system, is determined with the extracted portion of the address. *See, e.g., specification page 16, lines 5-15; page 19, lines 25-30.* The data address is aggregated with other addresses based at least in part on the memory reference object. *See FIG. 10, reference numeral 1015; specification page 21, lines 20-30.*

Independent claim 47 is directed to a computer program product for profiling code encoded on one or more machine-readable physical storage media. The computer program identifies a valid instruction instance corresponding to a runtime event. *See FIG. 4, reference numerals 405, 409; specification, e.g., page 14, line 20 – page 15, line 15.* A data address is determined based on the identified valid instruction instance. *See, e.g., specification page 13, lines 25-30.* A memory reference object, including virtual addressable memory, is determined with the data address. *See, e.g., specification page 16, lines 5-15; page 19, lines 25-30.* A set of addresses, including the determined data address, are aggregated based at least in part on the memory reference object. *See FIG. 10, reference numeral 1015; specification page 21, lines 20-30.*

Independent claim 55 is directed towards an apparatus including a processor, memory and means for identifying a memory reference object (including virtually addressable memory) and identifying a data address corresponding thereto from an instruction instance that corresponds to one of more runtime events. The "means for identifying a memory reference object and identifying a data address corresponding thereto from an instruction instance that corresponds to one or more runtime events" includes block 313 in FIG. 3. *See, e.g., specification page 12, lines 14-24.* In block 313 the runtime event is attributed to the identified source-level data object language construct. *See, e.g., specification page 13, lines 2-10, 15 -20 and 25-30.* As well as means for aggregating a set

of addresses that include the data addresses, based at least in part on the memory reference object. *See FIG. 10, reference numeral 1015; specification page 21, lines 20-30.*

VI.   GROUNDS OF REJECTION PRESENTED FOR REVIEW

A.   Whether independent claims 1, 16, 33, 41, 47, and 55, (and related dependent claims) are unpatentable under 35 U.S.C. § 102(e) over U.S. Patent No. 7,111,290 to Yates, Jr. et al. (hereafter "Yates").

VII.   ARGUMENT

A. THE REJECTION OF INDEPENDENT CLAIMS 1, 16, 33, 41, 47 and 55 AND THE RELATED DEPENDENT CLAIMS UNDER 35 U.S.C. § 102(e) IS IMPROPER BECAUSE THE YATES FAILS TO DISCLOSE VIRTUALLY ADDRESSABLE MEMORY OBJECTS

Claims 1-3, 5-12, 14-27, 29-37, 38-42, 44-50, 52-56 and 58 stand rejected under 35 U.S.C. § 102(e) as allegedly being unpatentable over U.S. Patent No. 7,111,290 to Yates Jr., et al. (hereafter "Yates"). Claims 1, 16, 33, 41, 47 and 55 are independent claims and the remaining claims depend therefrom. The basis of rejection of the dependent claims stems from the rejection of the independent claims; accordingly, the rejection of independent claims 1, 16, 33, 41, 47 and 55 is considered first.

A claim is anticipated only if each and every element as set forth in the claim is found, either expressly or inherently described, in a single prior art reference." *Verdegaal Bros. v. Union Oil Co. of California*, 814 F.2d 628, 631, 2 USPQ2d 1051, 1053 (Fed. Cir. 1987) (emphasis added). "The identical invention must be shown in as complete detail as is contained in the ... claim." *Richardson v. Suzuki Motor Co.*, 868 F.2d 1226, 1236, 9 USPQ2d 1913, 1920 (Fed. Cir. 1989) (emphasis added). Appellant respectfully suggests Yates fails to teach or suggest all of the limitations of independent claims 1, 16, 33, 41, 47 and 55, thus, Yates does not anticipate the claims.

Independent claim 1 includes the limitation that the profiling tool identifies "one or more memory reference objects" and the "memory reference objects include virtually addressable memory." The other independent claims include similar limitations, i.e. a profiling tool that profiles virtually addressable memory objects. Appellant respectfully submits that Yates does not teach this claim element. In fact, Yates specifically teaches the opposite.

Yates is generally directed to a system (referred therein as "Tapestry"), which may be programmed with non-Tapestry instructions (such as the Intel X86 instruction set architecture). The only teaching of code profiling in Yates occurs in Section V. titled, "Profiling to Determine Hot Spots for Translation," where the system appears to track often executed code segments. *See Yates col. 55, lines 5-30.* Yates describes how the Tapestry system profiles non-Tapestry code (i.e., X86 code), "profiler 400 monitors the execution of programs executing in X86 mode, and stores a stream of data representing the profile of the execution." *See Yates, col. 55, lines 7-10.* Yates states that the profiler tracks events "by physical address, rather than virtual address." *See Yates, col. 55, lines 47-48* (emphasis added). Other sections of Yates confirm that its profiling is not performed using virtually addressable memory objects, "Tapestry and TAXi operate in terms of the physical memory of the virtual X86, not the virtual or linear addresses." *See Yates, col. 29, lines 7-9*

(emphasis added). Yates even goes so far as to discuss the supposed benefits of not profiling code using virtual memory addresses. For example, by not profiling virtual memory addresses, Yates' code profiler does not have to "account for different virtual memory polices, process or thread management, or mappings between logical and physical resources, and [Tapestry can work] without any need to modify the operating system." *See Yates col. 55, lines 59-67.* Accordingly, Yates does not teach or suggest profiling using virtually addressable memory objects as required by claim 1.

Since all of the independent claims require code profiling using virtual memory objects, for at least the reasons recited above with respect to claim 1, it is respectfully submitted that Yates fails to disclose all of the limitations of independent claims 16, 33, 41, 47 and 55, and thus these claims are patentable under 35 U.S.C. § 102 over Yates.

All additional rejections to dependent claims 2-3, 5-12, 14, 15, 17-27, 29-32, 34-37, 39-40, 42, 44-46, 48-50, 52-54, 56, and 58 are based on Yates. The dependent claims include all of the limitations of the independent claims. Accordingly, it is respectfully submitted that these dependent claims are patentable over Yates for at least the same reasons as independent claims, 1, 16, 33, 41, 47 and 55.
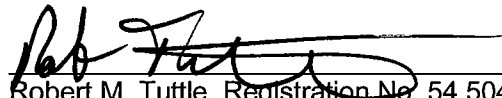
VIII. CONCLUSION

Appellant respectfully submits that all the appealed claims in this application are patentable and requests that the Board of Patent Appeals and Interferences direct allowance of the rejected claims.

Appellant believes no further fees or petitions are due with this filing. However, if any such petitions or fees are necessary, please consider this a request therefor and authorization to charge Deposit Account No. 04-1415 accordingly.

Dated: **4·28·2010**

Respectfully submitted,

Robert M. Tuttle, Registration No. 54,504
Attorney for Applicant
USPTO Customer No. 66083

DORSEY & WHITNEY LLP
Republic Plaza Building, Suite 4700
370 Seventeenth Street
Denver, Colorado 80202-5647
Phone: (303) 629-3400
Fax: (303) 629-3450

IX. APPENDIX OF CLAIMS

1.   A tangible computer readable storage medium comprising a computer-implemented software profiling tool that determines at least one data address from one or more instruction instances, and that identifies one or more memory reference objects, which are associated with the data address, as hindering execution of code that includes the instruction instances, wherein the instruction instances correspond to the code execution hindrance, wherein the memory reference objects include virtually addressable memory.

2.   The software profiling tool of claim 1 wherein the memory reference objects include one or more of physical memory reference objects and logical memory reference objects.

3.   The software profiling tool of claim 2 wherein the memory reference objects include one or more of a cache, cache lines, cache levels, cache sub-blocks, memory controllers, addressable memory, and memory-management page translation units.

4.   (Cancelled)

5.   The software profiling tool of claim 2 wherein the logical memory reference objects include one or more of source-level data objects, memory segments, heap variables, variable instances, and stack variables.

6.   The software profiling tool of claim 5 wherein the source-level data objects include one or more of functions, statically linked objects, data structures, data types, data type definitions, operands, and expressions.

7.   The software profiling tool of claim 6 wherein the statically linked objects include one or more of global variables and static variables.

8.   The software profiling tool of claim 1 wherein the software tool includes one or more of a compiler, an interpreter, an optimization tool, and a virtual machine.

9.   The software profiling tool of claim 1 wherein the code includes one or more of machine code, byte code, and interpreted code.

10. The software profiling tool of claim 1 that also aggregates addresses based on the memory reference objects.

11. The software profiling tool of claim 10 wherein the software tool utilizes at least a portion of the data addresses to aggregate the addresses.

12. The software profiling tool of claim 10 that also provides the aggregated addresses and an indication of the code execution hindrance corresponding to the aggregated addresses for one or more of storage and display.

13. (Cancelled)

14. The software profiling tool of claim 1 wherein the code execution hindrance corresponds to one or more sampled runtime events.

15. The software profiling tool of claim 14 wherein the sampled runtime events include one or more of cache misses, cache references, data translation buffer misses, data translation buffer references, and counter condition events.

16. A method for profiling code executing in a computer system, the method comprising:
    identifying an instruction instance that corresponds to a runtime event;
    determining a data address from the instruction instance; and
    determining a memory reference object from the determined address;
    wherein the data address includes a virtual address in the computer system.

17. The method of claim 16 wherein the runtime event is a sampled runtime event.

18. The method of claim 16 wherein identifying the instruction instance comprises backtracking from a second instruction instance to the instruction instance.

19. The method of claim 16 wherein determining the address from the instruction instance comprises decoding the instruction instance.

20. The method of claim 19 further comprising:

decoding the instruction instance if a register that hosts the instruction instance is determined as valid.

21. The method of claim 20 wherein determining if the register is valid comprises:
applying reverse register transformation with respect to the runtime event; and
determining whether the register is valid based on the applied reverse register transformation.

22. The method of claim 16 wherein the memory reference object includes a physical memory reference object or a logical memory reference object.

23. The method of claim 22 wherein the physical memory reference object includes cache, a cache line, a cache sub-block, a cache level, a memory controller, or a memory-management page translation unit.

24. The method of claim 16 wherein the logical memory reference object includes a source-level data object, a memory segment, a heap variable, or a stack variable.

25. The method of claim 24 wherein the source-level data object includes a data type, a data type definition, a statically linked object, an operand, a data structure, or an expression.

26. The method of claim 25 wherein the statically linked object includes a global variable or a static variable.

27. The method of claim 16 wherein the instruction instances include memory accessing instructions.

28. (Cancelled)

29. The method of claim 16 further comprising aggregating a plurality of addresses that include the determined address, based on the memory reference object.

30. The method of claim 29 wherein the aggregating of the plurality of addresses utilizes at least a portion of the addresses.

31. The method of claim 29 further comprising providing the aggregated plurality of addresses for one or more of display, storage, and manipulation.

32. The method of claim 16 embodied as a computer program product encoded on one or more machine-readable physical storage media.

33. A method of profiling code executing in a computer system, the method comprising:

associating data addresses with memory reference objects, wherein the data addresses have been determined from instruction instances corresponding to code execution hindrance; and

aggregating the data addresses based on their associated memory reference objects;

wherein the data addresses include virtual addresses in the computer system.

34. The method of claim 33 wherein the instruction instances include memory accessing instructions.

35. The method of claim 33 wherein the code execution hindrance corresponds to one or more runtime events.

36. The method of claim 35 wherein the runtime events are sampled runtime events.

37. The method of claim 35 wherein the runtime events include one or more of counter condition events, cache misses, cache references, data translation buffer references, and data translation buffer misses.

38. (Cancelled)

39. The method of claim 33 wherein said aggregating utilizes at least a portion of the data addresses.

40. The method of claim 33 embodied as a computer program product encoded on one or more machine-readable physical storage media.

41. A method of profiling code in a computer system comprising:

identifying an instruction instance corresponding to a runtime event;

determining whether the instruction instance is valid;

decoding the instruction instance to extract at least a portion of a data address if the instruction instance is valid;

determining a memory reference object with the extracted portion of the address; and

aggregating the data address with other addresses based at least in part on the memory reference object, wherein the memory reference object includes virtual addresses in the computer system.

42.   The method of claim 41 further comprising associating the extracted portion of the data address with the memory reference object.

43.   (Cancelled)

44.   The method of claim 41 wherein the runtime event is a sampled runtime event.

45.   The method of claim 41 further comprising:
applying reverse register transformation with respect to the runtime event to determine if the instruction instance is valid.

46.   The method of claim 41 embodied as a computer program product encoded on one or more machine-readable physical storage media.

47.   A computer program product for profiling code, encoded on one or more machine-readable physical storage media, the computer program product, which when executed, performs operations comprising:

identifying a valid instruction instance that corresponds to a runtime event;

determining a data address from the identified valid instruction instance;

determining a memory reference object with the determined data address; and

aggregating a set of addresses, which include the determined data address, based at least in part on the memory reference object, wherein the memory reference objects include virtually addressable memory.

48.   The computer program product of claim 47 wherein the operations further comprise associating the determined data address with the memory reference object.

49. The computer program product of claim 47 wherein the operations further comprise:

applying reverse register transformation with respect to the runtime event; and determining if the instruction instance is valid from the applied reverse register transformation.

50. The computer program product of claim 47 wherein the memory reference object includes a physical memory reference object or a logical memory reference object.

51. (Cancelled)

52. The computer program product of claim 50 wherein the logical memory reference object includes a source-level data object, a memory segment, a heap variable, a variable instance, and a stack variable.

53. The computer program product of claim 52 wherein the source-level data object includes a data type, a data type definition, an operand, a statically linked object, a data structure, or an expression.

54. The computer program product of claim 53 wherein the statically linked object includes a global variable or a static variable.

55. An apparatus comprising:
a processor;
memory; and
means for identifying a memory reference object and identifying a data address corresponding thereto from an instruction instance that corresponds to one or more runtime events, and aggregating a set of addresses that include the data address, based at least in part on the memory reference object, wherein the memory reference object includes virtually addressable memory.

56. The apparatus of claim 55 wherein the memory reference object includes a physical memory reference object or a logical memory reference object.

57. (Cancelled)

58. The apparatus of claim 56 wherein the processor includes event condition counters.

X.   **EVIDENCE APPENDIX**

None.

XI.   <u>RELATED PROCEEDINGS APPENDIX</u>

None.